

01 Jan 1993

Feasibility Test Verification Method For Nonsymmetric Release Time Task Scheduling

R. G. Karl

T. L. Lo

Daniel C. St. Clair

Missouri University of Science and Technology

Follow this and additional works at: https://scholarsmine.mst.edu/math_stat_facwork



Part of the [Mathematics Commons](#), and the [Statistics and Probability Commons](#)

Recommended Citation

R. G. Karl et al., "Feasibility Test Verification Method For Nonsymmetric Release Time Task Scheduling," *Proceedings - 26th Annual Simulation Symposium, ANSS 1993*, pp. 296 - 305, article no. 639151, Institute of Electrical and Electronics Engineers, Jan 1993.

The definitive version is available at <https://doi.org/10.1109/SIMSYM.1993.639151>

This Article - Conference proceedings is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Mathematics and Statistics Faculty Research & Creative Works by an authorized administrator of Scholars' Mine. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact scholarsmine@mst.edu.

Feasibility Test Verification Method for Nonsymmetric Release Time Task Scheduling

Randall G. Karl
Electronics & Space Corporation
St. Louis, Missouri

T. Leo Lo
McDonnell Douglas AIS Company
St. Louis, Missouri

Daniel C. St. Clair
University of Missouri-Rolla
Engineering Education Center
E-mail c0567@umrvmb.umar.edu

Abstract: This paper discusses problems associated with scheduling periodic tasks on a uniprocessor in a hard, real-time processing environment using a static-priority, preemptive-resume operating system. Task sets containing a single periodic task with two fixed release periods of unequal length are examined. The nonsymmetric scheduling algorithm is based on the rate monotonic scheduling algorithm which assigns higher task priorities to tasks with shorter release periods. The effects on processor utilization using two different priority assignment schemes are examined; one with task priorities sorted by the average release periods and the other with priorities sorted using the short nonsymmetric task period with the average period lengths for the remaining tasks. Results indicate that the second scheme had higher breakdown utilizations than the first one, and that the second scheme, for task sets with a low utilization nonsymmetric task, results in little or no loss in the overall task set utilization.

1. Introduction

Real-time systems must be designed to operate deterministically for time critical applications. These systems often consist of a set of periodic tasks which are released at regular intervals and have hard deadlines associated with their completion. Periodic tasks are useful for applications requiring control loops, gathering sensor data, or handling synchronous events. Scheduling tasks using non-preemptive scheduling algorithms and preemptive-resume scheduling algorithms in a hard, real-time environment have been widely researched [1, 4, 14, 20]. Non-preemptive methods employ a cyclic executive built into the program or control structure to manage the execution of periodic processes. Each periodic process is allowed to complete prior to relinquishing the processor to the controlling program. Preemptive-resume methods employ a priority based, event-driven dispatcher to schedule tasks. Tasks with lower priorities are removed from the processor when a

higher-priority task becomes ready for execution. At the completion of the higher-priority task, the lower-priority task resumes at the point it was preempted [5, 6].

Scheduling research has focused on periodic task sets with symmetric release times, where periodic events occur at fixed intervals of equal length. A periodic task set having a single task with periodic, nonsymmetric release times and hard deadlines may occur in some applications. In this instance, nonsymmetric release times are defined as the periodic intervals which consist of two consecutive fixed intervals of unequal length.

A simple and efficient algorithm has been developed to provide scheduling for applications with task sets containing n tasks; where a single task within the task set has nonsymmetric release times. This algorithm, called the nonsymmetric scheduling (NS) algorithm, provides scheduling for these applications when using static-priority, preemptive-resume scheduling. A priori knowledge of the task set execution times and periods is used by the algorithm to guarantee task set feasibility. Task set feasibility occurs when all scheduling deadlines are met.

The NS algorithm is tested against two priority assignment methods based on the rate monotonic algorithm. The rate monotonic algorithm assigns higher priorities to tasks with shorter periods. The first nonsymmetric, task-priority assignment method assigns higher priorities to tasks with shorter, average release periods. The second nonsymmetric, task-priority assignment method assigns priorities using the short, nonsymmetric task period with the remaining tasks using their average periods.

In Section 2, related research in real-time scheduling algorithms is reviewed. Non-preemptive time division multiplexing scheduling is presented along with two preemptive-resume scheduling algorithms; the earliest

deadline and rate monotonic algorithms. Section 3 introduces some basic notation and compares the proposed NS algorithm with the rate monotonic algorithm. In Section 4, the simulation method for characterizing the NS algorithm is discussed. The evaluation and analysis of the simulation results are presented in Section 5. Task set utilizations are provided to show the effects of the two nonsymmetric task priority assignment methods. Problems and limitations of the nonsymmetric tasks are discussed. Research results are summarized in Section 6.

2. Related Work

A number of researchers have studied the problem of scheduling tasks in a real-time environment. Methods may be divided into scheduling tasks which use non-preemptive scheduling algorithms and methods using preemptive-resume scheduling algorithms.

Non-preemptive scheduling algorithms are implemented using a cyclic executive [1]. One form of the cyclic executive uses time division multiplexing (TDM) with round robin scheduling. TDM assigns each task one or more timeslices within a minor cycle. A minor cycle is the basic unit of time for a cyclic executive and is defined as a fixed interval that a defined sequence of tasks or instruction streams must be executed. A major cycle consists of one or more minor cycles defined in a fixed order. The minor cycles within the major cycle definition are sequentially processed, returning to the initial minor cycle at the completion of the last minor cycle. The boundaries of the minor cycle provide points at which timing constraints are enforced. If the tasks executing within the minor cycle have not completed by the minor cycle boundary, an error called a frame overrun has occurred. If the tasks complete early, then an idle task or background task is executed. While this technique could be implemented to support nonsymmetric periods, there are too many practical problems to consider its use. For example, major and minor cycle timelines must be determined through repeated tuning trials to assure timing correctness. The technique lacks flexibility to meet changing needs or requirements during maintenance [3, 24]. Determining an optimal non-preemptive schedule for one processor is known to be NP-hard [4, 7].

The earliest deadline algorithm for dynamic-priority, preemptive-resume scheduling has been shown to be optimal for uniprocessor applications [14]. In practice, the earliest deadline algorithm suffers from several problems which may cause unpredictable scheduling under transient overloads [18]. For example, stochastic task execution times cause scheduling instability to occur when

the worst case execution time is much larger than the average execution time. If an overload develops, task sets scheduled using the earliest deadline algorithm will miss deadlines in a nondeterministic way. A task set is stable for a set of critical tasks, if and only if, all critical task deadlines are met even under an overload condition. The remaining tasks are considered non-critical, so that missing their deadlines during an overload condition will not cause a catastrophic failure. Reducing utilization to obtain stable scheduling may result in unreasonably low utilization. Recent research has focused on fault tolerant deadline mechanisms [2, 4]. These methods are not as efficient as static-priority, preemptive-resume scheduling.

The rate monotonic scheduling algorithm for static-priority, preemptive-resume operating systems has been shown to be optimal for uniprocessor applications [14, 22]. The rate monotonic algorithm assigns priorities by the task request rate with the highest request rate task receiving the highest priority and the lowest request rate task receiving the lowest priority. The rate monotonic algorithm is stable, since under worst case conditions only lower-priority tasks will miss their deadlines. When configuring the application, the designer must ensure that the mission critical tasks receive the higher priority assignments, while adhering to the rate monotonic algorithm's priority assignments. If a mission critical task has a low request rate relative to the other tasks within the application, the priority may be raised by using the period transform technique [18]. Lower-priority tasks may be used for background servicing, built-in self-test, or for tasks with soft deadlines. If there is some slack time associated with the completion of a task and the slack time is based on some statistical distribution of terminations, then the deadline is considered to be a soft deadline [8]. Research on the rate monotonic algorithm has provided scheduling enhancements for aperiodic tasks by using the priority exchange and deferrable server algorithms [12, 23]. Other enhancements include communication bus scheduling [11, 17, 21, 24] and static priority task synchronization via the priority ceiling protocol [9, 20].

3. Proposed Algorithm

This section reviews the fundamentals of the rate monotonic algorithm which will be required in subsequent discussions. A real-world example, where nonsymmetric periods occur, is presented. This is followed by the proposed NS algorithm.

A set of independent periodic tasks, τ , consists of n tasks, $\{\tau_1, \tau_2, \tau_3, \dots, \tau_n\}$. A task, τ_i consists of the 2-tuple (T_i, C_i) , where the integer i , $1 \leq i \leq n$, is the

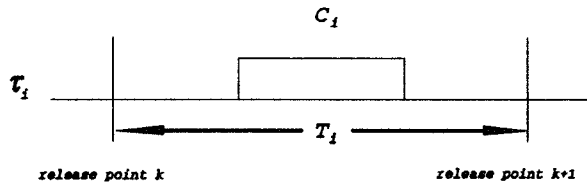


Figure 1. Release Period and Execution Cost for τ_i .

priority of τ_i , T_i is the task period and C_i is the execution cost of τ_i . The highest priority task is τ_1 with τ_n having the lowest priority. The task period is defined as the time from the release point k until the next release point, $k+1$ (see Figure 1). The deadline for τ_i is the next release point at the end of task period T_i . The execution cost C_i is the execution time required by τ_i independent of any preemption by other tasks. The value of C_i is constant and does not vary with time. For rate monotonic scheduling, static priorities are assigned according to the request period, so the task with the shortest period receives the highest priority and the task with the longest period the lowest priority. Processor utilization for τ_i is defined by C_i/T_i and the total processor utilization for all tasks by

$$U = \sum_{i=1}^n \frac{C_i}{T_i}.$$

Maximizing processor utilization is important provided the task set meets all deadlines. A task set is feasible, if and only if, all deadlines are met. A task set is infeasible if an increase in C_i , for $1 \leq i \leq n$, causes any task to miss a deadline. A task set utilizing the rate monotonic algorithm priority assignment is guaranteed to be feasible provided the utilization meets the following criteria [14]:

$$\sum_{i=1}^n \frac{C_i}{T_i} \leq n(2^{1/n} - 1).$$

As $n \rightarrow \infty$, the utilization converges toward 69.3% ($0.693 = \ln 2$). This utilization is referred to as the least upper bound and is the worst case schedulability bound. A task set may have higher utilization, but feasibility is not guaranteed by this feasibility test.

An improved feasibility test was created based on task release time phasing [19]. The task release time phasing defines when a task's release points occur relative to the task release points for other tasks within the task set. The critical instant is the point in time at which τ_i and all higher-priority tasks are released simultaneously. The worst case response time for τ_i will result from this phasing. The critical time zone for τ_i is the time from the critical instant until the completion of τ_i . The task τ_i must

complete prior to T_i or its deadline will be missed. If all tasks are released simultaneously and each task meets its first deadline, then all deadlines will be met for all other release time phasings. The following equation provides a method for checking the feasibility of a set of n independent tasks during the critical time zone by analyzing the scheduling points and the time used by each task during a scheduling period [19]:

$$\forall i, 1 \leq i \leq n, \min_{(k, q) \in R_i} \sum_{j=1}^i C_j \frac{1}{q T_k} \left\lceil \frac{q T_k}{T_j} \right\rceil \leq 1$$

where $R_i = \{(k, q) \mid 1 \leq k \leq i, q = 1, \dots, \lfloor T_i/T_k \rfloor\}$. The notation $\lceil z \rceil$ and $\lfloor z \rfloor$ represent the upper and lower bounds of z , respectively.

The feasibility test accounts for the scheduling of higher-priority tasks which may block τ_i by preemption and for the execution cost of τ_i . The test iteratively checks the scheduling points and execution costs for task τ_i and the higher-priority tasks τ_k which may affect the feasibility of τ_i . The term q is a range of values from 1 to $\lfloor T_i/T_k \rfloor$, which represents the number of schedule points τ_k has during T_i . A task τ_i is considered feasible, if it meets the inequality for at least one schedule phasing. The average utilization bound for a group of randomly generated task sets is approximately 88% [13] and 100% for task sets with uniformly harmonic tasks periods [19]. The rate monotonic algorithm does not provide for the possibility of a single task having periodic nonsymmetric release times. A nonsymmetric task is defined as having two constant task periods of unequal duration.

In real-world applications, static-priority, preemptive-resume operating systems are used to control embedded systems. These systems typically have periodic events which occur at fixed intervals of equal lengths and are used to schedule tasks. However, in some applications, these fixed intervals will consist of two consecutive intervals of unequal length. For example, an application may interface with a MIL-STD-1553 communications bus.

Messages received by the embedded system over a MIL-STD-1553 communications bus occur periodically, but the interval at which the messages are received is derived from the message positions within the overall message traffic set [16]. Message traffic is defined in terms of major and minor cycles [15]. A minor cycle defines a group of messages to be issued at specific times within a given period of time. A major cycle consists of one or more minor cycles processed in a fixed order with relatively fixed timing. Therefore, messages arrive

periodically, but the interarrival periods may have unequal, fixed length periods, as indicated by T_{long} and T_{short} in Figure 2. The bus controller acts as a bus master and requests information from remote terminals. The remote terminals must respond within a specific time.

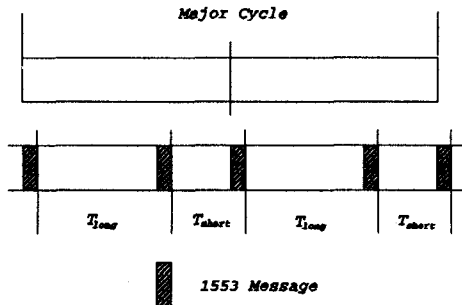


Figure 2. MIL-STD-1553 Message Traffic Example.

Message data received by a remote terminal may be used to control the position of a radar antenna, perform navigational updates or provide data for other activities needed to allow the system to operate correctly. It is important that the remote terminal responds to and processes the periodic messages prior to the arrival of a new set of messages, since sufficient message buffering hardware may not be available. Message arrivals are used as interrupt events to request execution of the message handling task. Scheduling the software to support periodic message traffic in a deterministic manner is necessary regardless of the time intervals in which the messages are received. Use of the period transform algorithm may assist scheduling, provided sufficient interrupt hardware support is available and the application can be successfully partitioned. However, the proposed NS algorithm, enhances the extended feasibility test to include instances of a task set containing n tasks with a single task having nonsymmetric release times.

The extended feasibility test, discussed previously, defined task sets as a 2-tuple, (T_i, C_i) , where T_i represented the task period and C_i defined the execution cost of τ_i . The NS algorithm supports the scheduling of task sets containing a single task with two task periods of unequal length. A task period adjustment constant, α , has been added to the extended feasibility test to provide a mathematical means of defining the two unequal length periods. If $\alpha = 0$ for all tasks defined within the task set, then the NS algorithm becomes the functional equivalent of the extended feasibility test.

In the NS algorithm, a set of independent periodic tasks, τ consists of n tasks, $\{\tau_1, \tau_2, \tau_3, \dots, \tau_n\}$. A task, τ_i

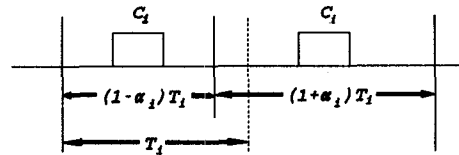


Figure 3. General Nonsymmetric Task Period Definition.

consists of the 3-tuple (T_i, C_i, α_i) , where T_i is the average task period. The real constant α_i , $0 \leq \alpha_i \leq 1$, is used to adjust the release periods of τ_i . For task set τ , let τ_m be the single nonsymmetric task such that $\tau_m = (T_m, C_m, \alpha_m)$ and $\tau_i = (T_i, C_i, 0)$ for $i \neq m$. The unequal length periods for τ_m are defined as, $T_{short} = (1 - \alpha_m)T_m$ and $T_{long} = (1 + \alpha_m)T_m$. The value of C_m is fixed. The deadline for τ_m is the end of the currently active period from $\{T_{short}, T_{long}\}$. Figure 3 shows the general nonsymmetric task definition for τ_i .

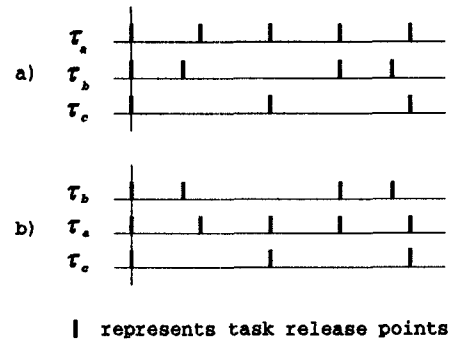


Figure 4. Task Priority Assignment Methods.

Two static priority assignments are possible for τ_m ; use the average task period T_m or use the short period T_{short} . The remaining tasks are assigned priorities by the rate monotonic algorithm. For example, let the task set consist of three tasks, $\tau_a = (100, 40, 0.0)$, $\tau_b = (150, 30, 0.5)$, and $\tau_c = (200, 30, 0.0)$. The nonsymmetric task periods for τ_b are $T_{short} = (1 - 0.5)150 = 75$ and $T_{long} = (1 + 0.5)150 = 225$. If the average task period priority assignment is used, then the tasks listed in priority order would be τ_a, τ_b , and τ_c , since $100 < 150 < 200$, where τ_a is the highest priority task (see Figure 4a). If the short period priority assignment is used, then the tasks listed in priority order would be τ_b, τ_a , and τ_c , since $75 < 100 < 200$, where τ_b is the highest priority task (see Figure 4b).

The NS algorithm has been partitioned into two sections. The iterative portion of the NS algorithm is provided in

Figure 5 and the feasibility test portion of the algorithm presented in Figure 6. The iterative portion of the algorithm uses the nonsymmetric feasibility test. For the task set to be feasible, each task is inserted, one at a time, by priority order into the task set and a feasibility test performed. If the algorithm indicates the task set is not feasible, then the utilization must be reduced. Otherwise, the task set is feasible and another task may be added. This is repeated until all tasks have been successfully added to the task set. The algorithm assumes that the task set, τ , was previously sorted in the desired priority order.

```

procedure Nonsymmetric_Scheduling_Algorithm is
 $\tau \equiv \{\tau_1, \dots, \tau_n\}$ ,  $i := 1$ 
loop
  if NS_Feasibility_Test( $i$ ) = true then
    if  $i \neq n$  then  $i := i + 1$ 
    else task set is feasible, exit
    end if
  else task set is not feasible, reduce utilization or exit
  end if
end loop
end Nonsymmetric_Scheduling_Algorithm

```

Figure 5. Nonsymmetric Scheduling Algorithm.

The nonsymmetric feasibility test portion of the NS algorithm is listed in Figure 6. The NS algorithm enhances the extended feasibility test by allowing the task set to contain a single task with nonsymmetric release periods. The release point for the nonsymmetric task short period, T_{short} , is set to occur at the critical instant. The feasibility for τ_m is determined using T_{short} as the critical time zone, since this provides the worst case, task-period phasing for τ_m . The remaining tasks have symmetric periods, therefore the worst case phasing for these tasks is known to occur at the critical instant [14]. The NS algorithm uses this as the basis for the nonsymmetric feasibility test. The NS algorithm may also be used to determine the feasibility for symmetric task sets, where $\alpha_i = 0$ for i , $0 \leq i \leq n$. In this instance, the NS algorithm is functionally equivalent to the extended feasibility test.

4. Empirical Testing

Testing methods were developed to investigate the NS algorithm. Two priority assignment methods were employed to investigate the impact on processor utilization caused by the nonsymmetric release times. A preemptive-resume operating system simulation was developed to validate the scheduling results and provide data to evaluate the characteristics of the nonsymmetric release times on

```

function NS_Feasibility_Test( $i$ ) returns Boolean
function Number_of_Schedule_Points( $i, k$ ) returns  $q$ 
 $q := 0$ ,  $Sum\_T_k := (1 - \alpha_k)T_k$ 
while  $((1 - \alpha_i)T_i \geq Sum\_T_k)$ 
   $q := q + 1$ 
   $Sum\_T_k := Sum\_T_k + (1 - \alpha_k)T_k(q + 1 \bmod 2) +$ 
     $(1 + \alpha_k)T_k(q \bmod 2)$ 
end while
end Number_of_Schedule_Points
function Scheduling_Test_Period( $q, k$ ) returns  $P_k$ 
if  $\alpha_k = 0.0$  then  $P_k := qT_k$ 

else  $P_k := \sum_{p=1}^q (1 - \alpha_k)T_k(p \bmod 2) +$ 
   $(1 + \alpha_k)T_k(p + 1 \bmod 2)$ 
end Scheduling_Test_Period
function Schedule_Pts_in_Period( $P_k, j$ ) returns  $x$ 
 $x := 1$ ,  $Sum\_T_j := (1 - \alpha_j)T_j$ 
while  $(P_k > Sum\_T_j)$ 
   $x := x + 1$ 
   $Sum\_T_j := (1 - \alpha_j)T_j(x \bmod 2) +$ 
     $(1 + \alpha_j)T_j(x + 1 \bmod 2)$ 
end while
end Schedule_Pts_in_Period
for ( $k$  in  $1..i$ )
   $q\_limit :=$  Number_of_Schedule_Points( $i, k$ )
  for ( $q := 1$  while  $q \leq q\_limit$  by 1)
     $C_{\text{total}} := 0$ ,  $P_k :=$  Scheduling_Test_Period( $q, k$ )
    for ( $j$  in  $1..i$ )
       $C_{\text{total}} := C_{\text{total}} + (C_j)(\text{Schedule\_Pts\_in\_Period}(P_k, j))$ 
    end for
    if  $C_{\text{total}} \leq P_k$  then return true
  end for
end for
return false
end NS_Feasibility_Test

```

Figure 6. Nonsymmetric Feasibility Test Algorithm.

scheduling. Comparisons of the data produced for the two priority assignments were made.

The simulations were based on a simple model of a preemptive-resume operating system which could provide information on the results of the NS algorithm. The simulation measured task latency and response times, the number of release requests and number of deadline overruns. Many parameters were varied to characterize the effects of the nonsymmetric periods. The utilization for each task set was iteratively lowered until the NS algorithm indicated that the task set was feasible. This is defined as the breakdown utilization, since any utilization level above this value will cause a task to miss its deadline. A simulation was executed to ascertain the

validity of the feasibility test. The task set utilization was then increased by a single utilization step. A second simulation was performed on the increased utilization task set to ensure that a deadline overrun would occur.

Simulations were performed on selected task sets which exhibited different types of task utilization distributions. Simulations were performed for both sorting methods. In addition to determining the reliability of the NS algorithm, separate feasibility test runs were made without performing simulations on the task sets. These runs were used to characterize the average utilization for task sets containing from two to ten tasks. To investigate the effects of the two priority methods on processor utilization, a large number of task sets were produced. Task sets were created by generating uniformly distributed, pseudo random numbers for each task τ_i , which represented the average task periods T_i and a weighted value for the corresponding execution cost C_i . The pseudo random numbers were generated using the mixed congruential method [10]. Task periods T_i were selected such that, $\{T_i \mid \forall i, 9 \leq T_i \leq 200\}$ [23]. Execution costs C_i were selected such that, $\{C_i \mid \forall i, 1 \leq C_i \leq 200\}$. The task set was scaled using a scaling factor s and a weighting factor w to derive the desired dynamic range and utilization level, such that the task set $\tau \equiv \{(sT_1, swC_1, \alpha_1), \dots, (sT_n, swC_n, \alpha_n)\}$.

The task set size n was varied to investigate the effects of the nonsymmetric release times on processor utilization, where $2 \leq n \leq 10$. The amount of skew affecting the lengths of the two unequal periods was varied such that, $0.0 \leq \alpha_m \leq 0.9$, where m is the nonsymmetric task number. For each feasibility test, only one task τ_m was allowed to have nonsymmetric periods, where $1 \leq m \leq n$. Task set breakdown utilization levels derived by the NS algorithm for each task set configured to have all symmetric periods, for $\alpha_i = 0.0$ where $1 \leq i \leq n$, generated comparable results when compared with breakdown levels derived by the extended feasibility algorithm for the same task set. Feasibility tests were conducted on all task sets for all task priority levels using a limited amount of skew. Totals for utilizations for each task set skew, priority level and sorting method were gathered. Utilization averages were plotted.

5. Evaluation and Analysis

This section presents results which characterize the effects of the nonsymmetric periods on average task set breakdown utilization. Graphs of the results are presented for both priority assignment methods. Effects of the two priority assignment methods on utilization are discussed.

5.1 Average Period Priority Assignment Results

The results from the average period priority assignment are presented in Figures 7 and 8. Figure 7 presents the average breakdown utilization for a task set consisting of five tasks, $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$. In this figure, for example, alpha task 1 represents the average task set utilization for task sets, where $m = 1$, such that $0.0 \leq \alpha_1 \leq 0.9$ for τ_1 and $\alpha_i = 0$, for τ_i where $i = 2, 3, 4, 5$. For alpha task 2, the average task set utilization is plotted, where $m = 2$, such that $0.0 \leq \alpha_2 \leq 0.9$ for τ_2 and $\alpha_i = 0$, for τ_i where $i = 1, 3, 4, 5$. The remaining tasks in the graph are plotted similarly. This convention has been followed for all breakdown utilization graphs. The number of task sets used in the characterizations was 532 and 252 task sets for sets containing five and ten tasks, respectively.

In Figure 7, as α_m is increased, the average task set utilization decreases. For alpha task 1, as α_1 increases, T_{short} decreases. The initial loss in utilization occurs in some task sets due to an increase in C_1 moving into the critical region of lower-priority tasks, thereby increasing the blocking time for these instances. As T_{short} decreases, $T_{short} \rightarrow C_1$. To meet the deadline requirement, the overall utilization must decrease. Since the derived task set utilization is computed utilizing a weighting factor w , the overall task set utilization is reduced. A different scaling method may change the task breakdown utilization levels.

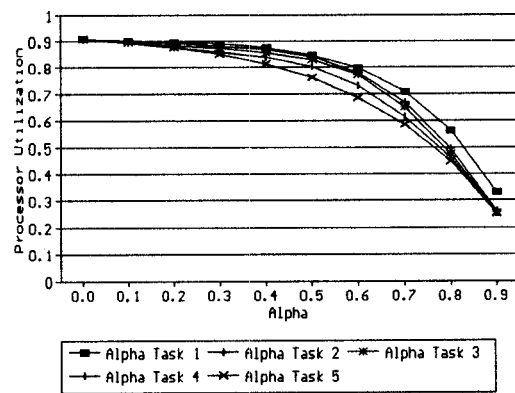


Figure 7. Average Breakdown Utilization for Task Sets with 5 Tasks using Average Period Priority Assignment.

The results for alpha task 2, 3, 4 and 5 are somewhat similar. However, as α_m increases, T_{short} becomes smaller than the periods of the higher-priority tasks. When this occurs, schedule points greater than T_{short} cannot be employed for scheduling τ_m , since they are beyond the deadline for τ_m . The loss of schedule points reduces the overall task set utilization in many instances. Additionally,

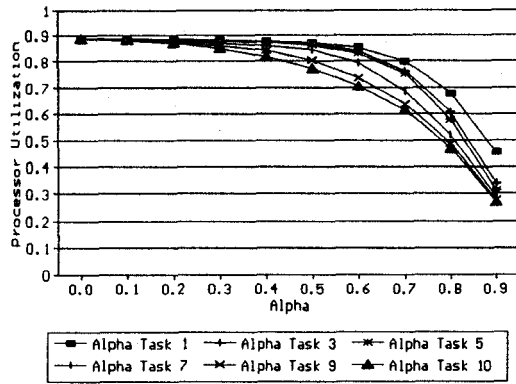


Figure 8. Average Breakdown Utilization for Task Sets with 10 Tasks using Average Period Priority Assignment.

as α_m increases, blocking by higher-priority tasks reduces the available slack time for C_m . Eventually,

$$T_{short} = \sum_{i=1}^m C_i.$$

Further reduction in T_{short} decreases the overall task set utilization. As m increases, the blocking becomes more severe and the loss of utilization more pronounced, as demonstrated by alpha task 5 in Figure 7. Figure 8 exhibits comparable results. As the number of tasks in the task set increases, the effects of the nonsymmetric period task decrease. This occurs since the utilization distribution among the tasks spreads out the effects.

5.2 Short Period Priority Assignment Results

The results from the short period priority assignment are presented in Figures 9 and 10. Figure 9 presents the average breakdown utilization for a task set consisting of five tasks, $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$. Results for alpha task 1 are equivalent to the average period priority assignment as shown in Figure 7. This result is expected, since the task priority for τ_1 cannot change.

The results for alpha task 2 indicate that raising the priority of τ_2 maintains the utilization level and results in a higher utilization than that of alpha task 1. Similar results are obtained for alpha tasks 3, 4 and 5. As α_m increases, T_{short} becomes smaller than the period of the next higher-priority task. When this occurs, the priority of τ_m is increased. This allows schedule points greater than T_{short} to be utilized when performing the feasibility test and differentiates the short priority assignment method from the average period priority assignment method in

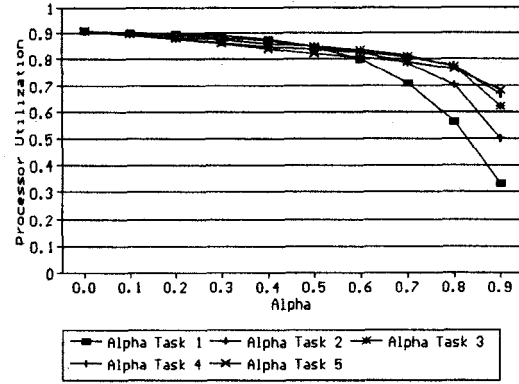


Figure 9. Average Breakdown Utilization for Task Sets with 5 Tasks using Short Period Priority Assignment.

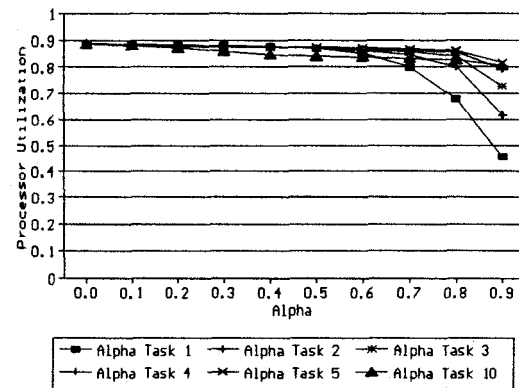


Figure 10. Average Breakdown Utilization for Task Sets with 10 Tasks using Short Period Priority Assignment.

this respect. The amount of blocking, caused by τ_m obtaining an increased priority level, is dependent upon its utilization C_m/T_m . Since $T_m > T_i$ for $m > i$, and the desired task set utilization is computed using weighting w , small reductions in the overall task set utilization will cause proportionally larger reductions in C_m relative to C_i .

5.3 Simulation Results and Analysis

Several task sets with different utilization distributions for individual tasks were selected for simulation from task sets generated for priority assignment characterizations. Figures 11 through 15 show individual task utilization distributions for symmetric task sets containing five tasks. Simulations were executed on task sets containing five tasks. For each nonsymmetric task m , α_m was varied from 0.1 to 0.9 in 0.1 steps. A simulation was also

provided for instances where all tasks have symmetric periods, that is, $\alpha_i = 0$ for all i . For example, a task set with five tasks would consist of forty-six individual simulations. Each simulation was executed two million clock ticks for each task at each α_m step.

The simulations yielded expected results. Feasible task set utilization levels derived from the NS algorithm were executed successfully with no deadline overruns. Increasing the utilization level for each simulation task set caused a deadline overrun to occur within the critical time zone. The deadline overrun location depended on the nonsymmetric task priority, utilization level, and value of α_m . Deadline overruns occurred earlier in the critical time zone as α_m was increased. Simulation results for the average period priority assignment method indicated that lower-priority nonsymmetric tasks with high utilization levels had lower breakdown utilizations throughout the range of α . Tasks with very low utilizations tended to

maintain their utilization levels until α approached 0.7. For $\alpha = 0.9$, utilizations converged towards 0.3 to 0.25.

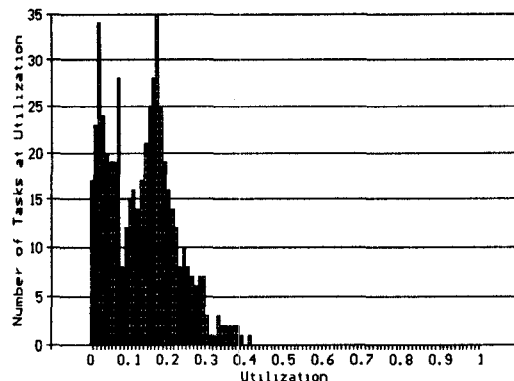


Figure 13. Symmetric Task 3 Utilization Distribution.

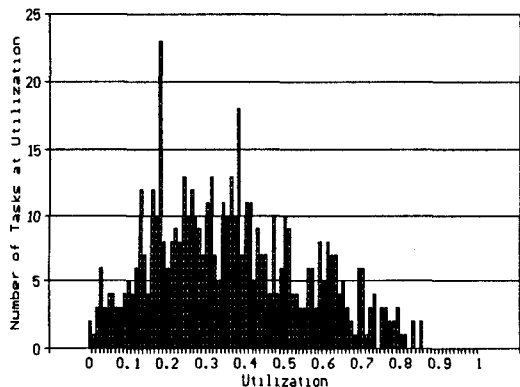


Figure 11. Symmetric Task 1 Utilization Distribution.

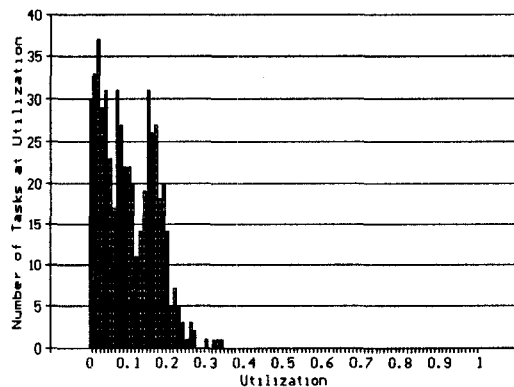


Figure 14. Symmetric Task 4 Utilization Distribution.

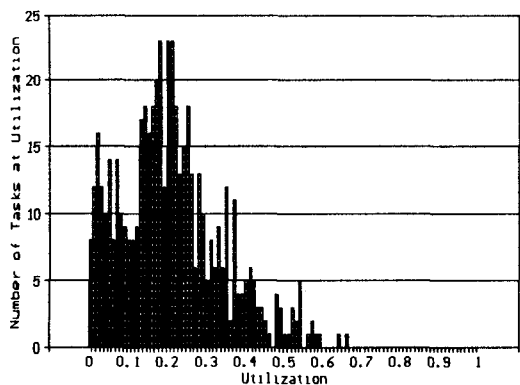


Figure 12. Symmetric Task 2 Utilization Distribution.

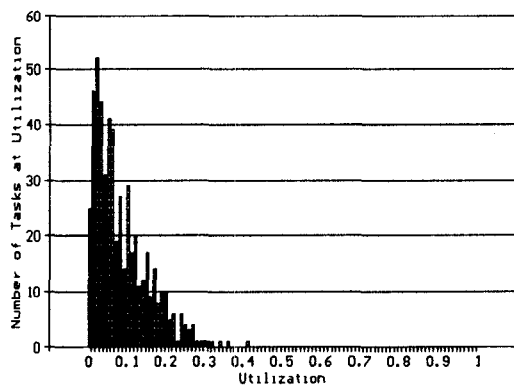


Figure 15. Symmetric Task 5 Utilization Distribution.

Short period priority assignment simulation results indicated that task sets with a low utilization nonsymmetric task, for example where $0.01 \leq U_m \leq 0.06$, lost little or no utilization throughout the range of α . Nonsymmetric tasks with high utilization had lower breakdown utilizations throughout the range of α .

Comparison of the two methods indicated that the short period priority assignment method yielded higher task set breakdown utilizations than the average period priority assignment method. Task set utilization for task sets with low utilization level nonsymmetric tasks benefitted the most from the short period priority assignment method. The breakdown utilization level for these task sets remained relatively constant throughout the range of α_m . Task sets with high utilization level nonsymmetric tasks also benefitted from using the short period priority assignment method. However, as α_m increased, the utilization losses were equivalent for both methods until the effects from raising the nonsymmetric task priority decreased the utilization loss. The raising of the task priority occurred for the short period priority assignment method when for the nonsymmetric task, T_{short} is less than the period of a higher-priority task.

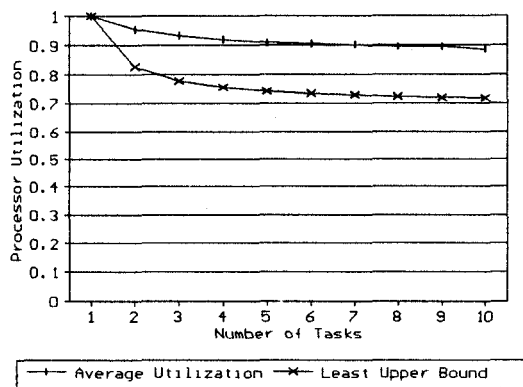


Figure 16. Average Symmetric Task Breakdown Utilization.

The NS algorithm provided functionally equivalent results for symmetric task sets when compared with the extended feasibility test. Figure 16 shows the average breakdown utilization for symmetric task sets. No differences were found when comparing the feasible task set utilization levels calculated by the two algorithms for the task sets generated for the priority assignment characterizations. Figure 16 also shows the rate monotonic least upper bound, which is the worst case

schedulability bound. This bound is very pessimistic when compared with the experimental average breakdown utilization.

6. Conclusion

The problems associated with scheduling periodic tasks in a hard, real-time environment using a static-priority, preemptive-resume operating system have been reviewed. The research has examined the scheduling problems associated with a task set containing a single periodic task which has nonsymmetric release times. A real-world application example was presented which may require task periodic release times, but whose tasking periods are not symmetric. The NS scheduling algorithm was proposed to schedule nonsymmetric task sets to satisfy their deadlines.

The NS algorithm enhances the extended feasibility test and provides an analytic method for determining the feasibility of a task set containing symmetric and nonsymmetric period tasks. A task period adjustment constant, α , was added to the extended feasibility test to provide a mathematical means of defining the two unequal length periods. If $\alpha = 0$ for all tasks defined within the task set, then the NS algorithm becomes the functional equivalent of the extended feasibility test. Both algorithms verify task set feasibility during the critical time zone. The release point of the nonsymmetric period task short period, T_{short} , is set to occur at the critical instant. The feasibility for τ_m is determined using T_{short} as the critical time zone, since this provides the worst case, task-period phasing for τ_m . The remaining tasks have symmetric periods, therefore the worst case phasing for these tasks is known to occur at the critical instant. The NS algorithm uses this as the basis for the nonsymmetric feasibility test.

The effects on processor utilization were examined using two different priority assignment schemes. Two static priority assignments are possible for τ_m , using the average period T_m or the short period T_{short} . The tasks are assigned priorities by the rate monotonic algorithm using the average periods for all symmetric tasks and either short or average periods for nonsymmetric task.

Simulations and task set breakdown utilization characterizations were executed on several different sizes of task sets. Simulations were used to verify the feasibility test portion of the NS algorithm. The task set breakdown utilization characterizations were used to determine the average case effects for the two nonsymmetric task priority assignment methods. The

average breakdown utilizations for the two methods were presented. Maximizing the processor utilization is desirable, provided the tasks operate in a deterministic manner and meet their deadlines. The average breakdown utilization results indicated that the short period priority assignment produced higher breakdown utilization levels than the average period priority assignment method. Nonsymmetric tasks with low utilizations benefitted from the short period priority assignment method.

Future research could be conducted to determine the effects of more than one nonsymmetric period task on the task set breakdown utilization. Different scaling methods could be used to modify the individual task utilization levels. The NS algorithm could also be extended to include blocking times caused by interprocess communication using server tasks.

Reference

- [1] Baker, T. P., Shaw, A., "The Cyclic Executive Model and Ada", *Proceedings of the Real-Time Systems Symposium*, IEEE, December, 1988, pp. 120-129.
- [2] Campbell, R. H., Horton, K. H., Belford, G. G., "Simulations of a Fault-Tolerant Deadline Mechanism", *9th Annual International Symposium on Fault-Tolerant Computing*, IEEE, June, 1979, pp. 95-101.
- [3] Carlow, G. D., "Architecture of the Space Shuttle Primary Avionics Software System", *Communications of the ACM*, Volume 27, Number 9, September 1984, pp. 926-936.
- [4] Chetto, H., Chetto, M., "Some Results of the Earliest Deadline Scheduling Algorithm", *IEEE Transactions on Software Engineering*, Volume 15, Number 10, October, 1989, pp. 1261-1269.
- [5] Coffman, E. G. Jr. (Ed.), *Computer and Job-Shop Scheduling Theory*, J. Wiley & Sons, NY, NY, 1976.
- [6] Conway, R. W., Maxwell, W. L., Miller, L. W., *Theory of Scheduling*, Addison-Wesley, Reading, MA., 1967.
- [7] Garey, M. R., Johnson, D. S., *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, CA, 1979.
- [8] Gonzalez, M. J. Jr., "Deterministic Processor Scheduling", *ACM Computing Surveys*, Volume 9, Number 3, Sept. 1977.
- [9] Goodenough, J. B., Sha, L., "The Priority Ceiling Protocol: A Method for Minimizing the Blocking of High-Priority Ada Tasks", Special Report CMU/SEI-88-SR-4, Carnegie-Mellon University, CS Department, March 1988.
- [10] Lavenberg, S. S., *Computer Performance Modeling Handbook*, Academic Press, New York, New York, 1983.
- [11] Lehoczky, J. P., Sha, L., "Performance of Real-Time Bus Scheduling Algorithms", *Performance '86 and ACM SIGMETRICS 1986*, May 28-30, 1986, pp. 44-53.
- [12] Lehoczky, J. P., Sha, L., Strosnider, J. K., "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", *Proceedings of the Real-Time Systems Symposium*, IEEE, December 1-3, 1987, pp. 261-270.
- [13] Lehoczky, J., Sha, L., Ding, Y., "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", *Proceedings of the Real-Time Systems Symposium*, IEEE, December, 1989, pp. 166-171.
- [14] Liu, C. L., Layland, J. W., "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the ACM*, Volume 20, Number 1, January 1973, pp. 46-61.
- [15] *MIL-STD-1553 Designer's Guide*, DDC ILC Data Device Corporation, Bohemia, New York, 1982.
- [16] *MIL-STD-1553B, Aircraft Internal Time Division Command/Response Multiplex Data Exchange Bus*, Department of Defense, Directorate of Material Management, Warner Robins Air Logistics Center, Warner Robins, Georgia, February 25, 1985, NOTICE 2.
- [17] Rajkumar, R., Sha, L., Lehoczky, J. P., "On Countering the Effects of Cycle-Stealing in a Hard Real-Time Environment", *Proceedings of the Real-Time Systems Symposium*, IEEE, December 1-3, 1987, pp. 2-11.
- [18] Sha, L., Lehoczky, J. P., Rajkumar, R., "Solutions for Some Practical Problems in Prioritized Preemptive Scheduling", *IEEE Real-Time Systems Symposium*, Dec., 1986, pp. 181-191.
- [19] Sha, L., Goodenough, J. B., "Real-Time Scheduling Theory and Ada", Technical Report CMU/SEI-89-TR-14, Carnegie-Mellon University, Computer Science Department, April, 1989.
- [20] Sha, L., Goodenough, J. B., "Real-Time Scheduling Theory and Ada", *Computer*, Volume 23, Number 4, April, 1990, pp. 53-62.
- [21] Sha, L., Rajkumar, R., Lehoczky, J. P., "Real-Time Scheduling Support in Futurebus+", *Proceedings of the Real-Time Systems Symposium*, IEEE, December, 1990, pp. 331-340.
- [22] Serlin, O., "Scheduling of time critical processes", *Proceedings of AFIPS 1972 Spring Joint Computer Conference*, AFIPS Press, Montvale, N.J., 1972, pp. 925-932.
- [23] Sprunt, B., Lehoczky, J., Sha, L., "Exploiting Unused Periodic Time for Aperiodic Service Using the Extended Priority Exchange Algorithm", *Proceedings of the Real-Time Systems Symposium*, IEEE, December 6-8, 1988, pp. 251-258.
- [24] Strosnider, J. K., Marchok, T., Lehoczky, J., "Advanced Real-Time Scheduling Using the IEEE 802.5 Token Ring", *Proceedings of the Real-Time Systems Symposium*, IEEE, December, 1988, pp. 42-52.